TARTU ÜLIKOOL

Loodus- ja tehnoloogiateaduskond

Tehnoloogiainstituut

Tarmo Kalling

JÄRJEPIDEVA VEEBIMONITOORIMISE SÜSTEEMI ARHITEKTUUR EESTI DOMEENIS

Bakalaureusetöö (12 EAP)

Juhendaja: Peep Küngas, Ph.D

Luban töö kaitsmisele:

Juhendaja …….

Programmijuht …….

Tartu 2015

Resümee

Internetis on andmemahud aastate jooksus eksponentsiaalselt kasvanud [1]. Pelgalt dokumendipõhise interneti kitsaskohad on hakanud välja lööma. Andmete mahud on niivõrd suureks läinud, et on tekkinud järjest kasvav vajadus lahenduste järele, mis ise jälgivad veebis toimuvat ning teavitavad huvitatud osapooli muutustest. Erinevaid selliseid süsteeme on loodud erinevate eesmärkide jaoks [3][4][5][9][10]. Käesolevas töös kirjutatakse esmalt erinevatest süsteemidest ja seletatakse, et eri põhjustel ükski neist hästi selle töö jaoks ei sobi. Enamus neist ei sobi sellepärast, et need ei ole lihtsalt seotavad teiste rakendustega, aga põhjuste hulka võib kuuluda ka kõrge hind, liigne spetsiifilisus või ligipääs lähtekoodile. Seetõttu komplekteeritakse ise süsteem, mis jälgib internetis toimuvat, kaardistab ajas tekkinud erinevused ning saadab need huvitatud osapooltele linkandmetena. Süsteem on mõeldud töötama Eesti domeeni peal ja seda testitakse siin töös paarisaja avaliku sektori veebilehe peal. Nii hinnatakse süsteemi erinevate osade töökindlust ning ka valitud strateegia andmete täielikkuse ja aja suhet. Viimaks analüüsitakse realiseeritud süsteemi kitsaskohti ja seda, kuidas neid elimineerida saaks.

Sisukord

Jooniste loetelu

Terminid

RDF (ingl Resource Description Framework) - W3C poolt väljastatud juhiste kogum, kuidas standardiseeritult kirjeldada metaandmeid. RDF/XML on moodus RDF standardile vastavalt metaandmeid XML kujul kirjeldada.

SPARQL (ingl SPARQL Protocol and RDF Query Language) - Päringu keel, mille abil saab infot pärida RDF kujul olevate andmete kohta.

C-SPARQL (ingl Continous SPARQL) - SPARQL edasiarendus, mis võimaldab päringuid teha reaalajas SPARQL voogude pealt.

API (ingl Application Programming Interface) - Valik tarkvara tootja poolt välja antavaid vahendeid toodetud tarkvaraga liidestamiseks.

WARC (ingl Web ARChive) - Arhiivi fail, mis sisaldab (potentsiaalselt erinevatest allikatest) veebilehtede sisu tõmmiseid koos nende juurde kuuluva metainfoga. Veebiroomaja tulemfail.

REST (Representational State Transfer) - Kogumik juhiseid arendamaks skaleeruvaid veebiteenuseid. Selliselt arendatud veebiteenust kutsutakse RESTful teenuseks. Tihti on selliselt arendatud teenuses iga rakenduse olemi kohta URL ja vastavaid olemeid saab lisada, kustutada, uuendada ja pärida läbi sama URL erinevate HTTP meetodite. Nii saab tagada teenuse liidese intuitiivsuse. RESTful teenuste puhul ei säilitata serveris päringutevahelist seisu (näiteks ei hoita andmeid HTTP sessioonis).

1. Sissejuhatus

Kui 20. sajandi teises pooles tänapäevase veebi alustehnoloogiaid välja arendati, siis ei osanud mitte keegi prognoosida, kui suuri andmemahte veeb paarikümne aasta pärast sisaldama hakkab. Sellest tulenevalt keskenduti dokumendipõhisele internetile, kus rõhk on andmetel, mida inimesed internetti laevad ja vähem tähelepanu pööratakse nende andmete formaadile, masinloetavusele ning andmetevahelistele seostele. Internet oli mõeldud inimestele dokumentide üles ja alla laadimiseks. Tänapäeva internetis on andmemahud aga niivõrd suureks kasvanud, et pelgalt dokumendipõhise interneti kitsaskohad on hakanud välja lööma. Internetis liikus 2007. aastal igakuiselt andmeid viie *exabyte*'i väärtuses ja 2010. aastal oli veebis liikuvate andmete maht juba 21 *exabyte*'i kuus. Antud näitaja kasvab eksponentsiaalselt [1].

Suur probleem on suvalisel ajahetkel, milleks tihtipeale on värskeim võimalik aeg, aga mis võib olla ka mõni minevikus olev ajahetk, internetis olevast infost huvitava info alamhulga kättesaamine. Probleemi levinud lahenduseks on saanud veebi roomamine (*web crawling*), mis kujutab endast mingi automaatmehhanismi poolt ette antud interneti lehtede sügavuti töötlemist ja töödeldud info indekseerimist selliselt, et sellest on hiljem võimalik infot osade kaupa kätte saada. Roomamiste tulemusel saadud indeksi pealt otsingute tegemiseks on mõnikord nende peale ehitatud veebipõhised otsingumootorid nagu Google, Yahoo!, Bing jpt. Roomamist teostatakse perioodiliselt ja niimoodi üritatakse saavutada võimalikult ajakohased otsingutulemused. Indeksitest info kätte saamine käib populaarsete otsingumootorite puhul nii, et tarbija pärib, mis teda huvitab ja saab sünkroonse vastuse. Mainitud otsingumootoritel on küll teinekord tehtud võimalused, et erinevate protokollide ja standardite abil saaks arendajad oma rakendustega neid siduda ja seeläbi otsingumootorite indeksite vastu programmatiliselt päringuid teha ning oma äranägemise järgi vastuseid töödelda, aga sellel lähenemisel on mitu miinust.

Algatuseks peab värskeima info saamiseks pidevalt uusi päringuid tegema ja nende tulemusi vastu eelmisi päringuid võrdlema. Puudub võimalus enda päring teenusepakkuja juures registreerida ja jääda asünkroonset vastust ootama, kohe kui midagi uut teenusepakkuja indeksisse tuleb. Teiseks tagastavad otsingumootorid üldjuhul huvipakkuvat infot sisaldava lehe

URLi, aga ei anna koos sellega täpsustavat infot, kuidas lehtede seest just huvipakkuvat osa kätte saada. See tähendab, et sellist lähenemist kasutades on väga palju potentsiaalselt ressursikulukat tööd vaja ära teha ikkagi info tarbija pool.

Kui info tarbija soovib erinevate võimaluste abil jälgida, kuidas veebis info muutub, siis tekib probleem, et potentsiaalselt väikese tulemi nimel on tal vaja väga suur andmemaht maha salvestada, et hilisemat tõmmist oleks millegi vastu võrrelda. Või kui andmed on väga volatiilsed, siis on võimalus neid iga päringu ajal sünkroonselt uuesti roomata. Seda saab muidugi rahuldavalt kiirelt teha ainult suhteliselt väikese algallika puhul. Selline lahendus sobib näiteks lennupiletite hindade jälgimisel internetis leiduvast allikast. [2]

Internetis leiduvast ja pidevalt täienevast andmemassist spetsiifiliselt määratletud alamhulga kohta info saamise motiivid võivad olla väga erinevad. Üks motiiv on näiteks konkurentsimonitooring. Iga ettevõtjat huvitab, millega tegelevad tema konkurendid. Mida kiiremini jõuab ettevõtjani info tema konkurentide tegevuse kohta, seda kiiremini saab ta vastavalt reageerida ja nii saab ta olla ise kokkuvõttes efektiivsem. Veel võib näiteks tuua valdkonnauuringud. Inimest võib näiteks huvitada, millist infot leidub eestikeelsetel internetilehtedel alustavate iduettevõtete kohta. Motiiv sellist infot leida võib peituda näiteks soovis seda infot kasutada investeerimisotsuste tegemisel. Sarnaseid näiteid ja kõiki muid analoogseid situatsioone ühendab see, et internet on küll sellise info leidmiseks väga hea koht, aga sealt kogu potentsiaalselt huvi pakkuva info kättesaamine võib osutuda ülimalt keeruliseks ning aega ja arvutusressurssi nõudvaks ülesandeks. Seonduvaid probleeme on tänapäeval mitmete erinevate initsiatiivide raames juba lahendatud. Nendes võimalustes, mis info otsimiseks juba olemas on keskendutakse pigem viimasena leitud infole, mitte aga info muutusele ajas.

Veebist huvipakkuva info otsimise lahendusi on erinevaid ja nendest kirjutatakse üksikasjalisemalt punktis 2. Nõrkuseks, mis olemasolevaid lahendusi suures osas ühendab on see, et puuduvad standardsed liidestamisvõimalused teiste süsteemidega. Osadel lahendustel on need võimalused olemas, aga ka nendel on erinevad probleemid, nagu näiteks teenuse hind, ligipääs puudumine koodile või siis need ei ole mõeldud niivõrd suuremahulise töö jaoks.

Käesolevas bakalaureusetöös kirjeldatakse lahendust mõnest sajast domeenist koosneva veebi pidevaks monitoorimiseks. Lahendust hinnatakse Eesti avaliku sektori veebi monitoorimise

näitel. Töös hinnatakse lahenduse käivitamiseks vajalikke parameetreid muudatuste leidmise efektiivsuse seisukohast. Veelgi täpsemalt, tuvastatakse optimaalne veebilehtede värskendamise intervall Eesti avaliku sektori veebi muudatuste varase avastamise võtmes.

Selles töös pakutav probleemi lahendus seisneb selles, et teostatakse järjestikusi roomamisi ette antud veebilehtedele. Neid roomamisi võrreldakse omavahel ning võrdluse tulemiks on RDF/XML voog, mis antakse edasi C-SPARQL mootorile ja sõnumiedastusmehhanismile, mis omakorda peavad arvet lõpptarbijate ja päringute üle ning hoolitsevad, et tulemid saadetaks lõpptarbija poolt antud lõpppunktidele. Peatükis 3 kirjeldatakse selle lahenduse arhitektuuri, täpsustatakse kasutatavate komponentide tööpõhimõtet, põhjendatakse komponentide valikuid ja nende omavahelist ühendamist, selgitatakse muutusandmete liikumist alates veebi roomamisest kuni lõpptarbija päringu vastuseni. Lisaks kirjeldatakse ka selle töö raames valmistatud lisarakendusi, mida kogu lahenduse sujuvaks tööks ja valideerimiseks kasutati. Peatükis 4 kirjeldatakse, kuidas lahendust valideeritakse, roomates valikut avaliku sektori veebilehti ja mõõtes tuvastatud erinevuste koguarvu, teostades võrdlemisi erinevate intervallidega, ning seejärel analüüsides eksperimendi tulemusi, vaadates, kas on mõistlik roomata tihemini või harvemini allikaks olevaid domeene. Mida ja kui tihti roomata ei ole üheselt vastatav ja triviaalne küsimus. Seda on palju erinevate teadlaste poolt uuritud ja erinevaid lahendusi välja pakutud. Selleks, et saavutada parem andmete täielikkus on vaja põhjalikumaid roomamisi, mis võtab kokkuvõttes rohkem aega. Seega on tihtipeale vaja leida kompromiss andmekvaliteedi ja ajakulu vahel. Käesolevas töös ei keskenduta parimate roomamisstrateegiate leidmisele, aga siiski mõõdetakse, milline on ajakulu ja täielikkuse suhe selle töö raames realiseeritud lahenduse puhul ja seeläbi üritatakse üles leida realisatsiooni kitsaskohad.

2. Valdkonna ülevaade

Nõudlus rakenduste järele, mis loovad lisaväärtust uut informatsiooni erinevatest allikatest kokku koondades järjest kasvab ja sellist vajadust rahuldavaid süsteeme on realiseeritud erinevate raamatukogude, riiklike institutsioonide ja ka eraasutuste poolt mitmel pool maailmas. Järgnevalt on pikemalt kirjutatud mõnest projektist, mis on välja kasvanud analoogsest

nõudlusest nagu käesoleva töö raames realiseeritud lahendus. Samuti on juttu tänapäevastest lahendustest, mis on vabalt saadaval, et huvitatud osapooled saaksid ise luua mehhanismi, mis ekstraheeriks veebist neile vajamineva info. Arutatud on selliste lahenduste positiivsete ja negatiivsete poolte üle. Lisaks on kirjutatud, milliseid valikuid ja millistel põhjustel tehti antud lahenduse realiseerimisel väljundandmete osas.

2.1 Huvipakkuvate andmete internetist leidmine

Näide valdkonnast, kus võib tähtsaks osutuda info kiire ja usaldusväärne kättesaadavus on avaliku sektori suhtlus rahvaga. Austraalias on Tasmaania ülikooli teadlased seda teemat uurinud ja realiseerinud ka lahenduse. Sealne probleem seisnes selles, et Tasmaania avalike teenuste registri veebileht ja Tasmaania veebileht pidid kajastama tähtsamaid ajakohaseid uudiseid piirkonna ettevõtete, teenuste jms kohta, aga vastavate veebilehtede haldajad ei suutnud käsitsi jälgida kõike potentsiaalselt huvi pakkuvat infot, mis piirkondlike institutsioonide veebidesse jõudis. Uurimuse tulemusena valmis lahendus, mis jälgis kohalike institutsioonide veebilehti ning teavitas nende kahe veebi administraatoreid potentsiaalsetest huvi pakkuvatest infokildudest. Kliendi poolt saavutati tänu sellisele lahendusele suur võit ajas, töötajate rahulolus, avalike andmete ajakohasuses ja kvaliteedis. Käesoleva töö käigus tehtava lahendusega ja austraallaste lahendusel on aga üks oluline erinevus. Nende lahendus ei kasutanud semantilisi veebitehnoloogiaid, millele põhinevad linkandmed, sest vastasel juhul oleks uue lahenduse integreerimine vanade süsteemidega olnud ebaotstarbekalt kulukas ja keeruline. [3]

Iraani teadlased avaldasid 2012. aastal publikatsiooni projekti kohta, mis jälgib blogides toimuvat. Nende töö motivatsioon baseerus teooriale, et just blogides liigub kõige huvitavam ja päevakajalisem informatsioon ning blogides toimuv peegeldab seega hästi populaarseid trende. See lahendus jälgis blogipositusi ja lisaks ka nende juurde kuuluvaid kommentaariumeid. Töö keskendus suuresti sellele, kuidas väärtusliku teabega blogid üles leida ja nendest vastav info kätte saada. Keerukaks tegi selle töö suur hulk hüljatud blogisid, rämpsblogisid, rohke slängikasutus tekstides jne. Kuna blogid oma iseloomult on väga spetsiifiline alamhulk lehti, siis arendati selle ülesande jaoks välja ka spetsiaalne roomaja. Roomaja põhimõte oli selles, et kui

viimati roomatud leht hinnati piisavalt huvitavaks, siis roomati mööda sellel olevaid linke edasi, vastasel juhul mitte. Ei ole olemas konstantseid headeks hinnatud algallikaid, millest alati pihta hakatakse vaid algoritm ise leiab ka järgmise töö algallikad. Kui allikas tunnistatakse potentsiaalselt huvitavaks, siis küsitakse tema RSS voog ja analüüsitakse seal olevate postituste kokkuvõtteid. Kui see hinnatakse ka huvitavaks, siis lisatakse leht kaalutud graafi. Kui kõik potentsiaalselt huvitavad blogid on analüüsitud, siis hakatakse koostatud kaalutud graafi alusel reaalset sisu alles läbima. Kahjuks ei ole antud publikatsioonis välja toodud, millisel kujul andmed muudatuste kohta hoiti ja kuidas neid lõppkasutajale presenteeriti. [4]

Hiina teadlased avaldasid 2009. aastal publikatsiooni projekti kohta, mis jälgib e-kaubanduses toimuvat. Nende lahenduses sisendiks oli hulk domeenide ja hulk huvipakkuvaid teemasid. Lehti käidi ükshaaval läbi ja igast lehest ekstraheeriti välja ainult reaalne tekstiline sisu, mis jaotati teemade järgi sektoriteks. Igale sektorile anti tähtsushinnang ja selle baasil sai ka igale lehele anda hinnangu, kui tihti seda vaja külastada on. Kõige tähtsamaks hinnatud lehed anti edasi teemade repositooriumile, mis keeletöötluse ja masinõppe algoritmide alusel selgitas välja, millised olid uued potentsiaalselt huvitavad teemad. Töö käigus arendati välja spetsiaalne e-kaubandusega seotud ontoloogia, mille alusel andmeid esitati. [5]

Initsiatiivi “Britain on the Web” raames teostatakse Briti Raamatukogu poolt Suurbritannias ingliskeelse veebi arhiveerimist 2002. aastast saadik [6]. Algne eesmärk oli kaardistada Suurbritannia veebis toimuvat aktiivsust ning selleks roomati perioodiliselt läbi 100 veebilehte. Projekt loeti edukaks ja pärast esialgse eesmärgi täitmist on selle mahtu kuni tänapäevani kasvatatud ning sinna on kaasatud ka rohkem osapooli. Alates 2004. aastast roomatud tulemitele saab ligi ka avalikust veebist [7]. Töö kirjutamise hetkel ei leidnud töö autor avalikku liidest, mille abil liidestada väliseid rakendusi antud arhiivi külge päringuid tegema. Vähemalt ametlikult ja avalikult välja pakututest oli ainus viis päringute tegemiseks vastmainitud avalik veeb. Ka ei leidunud kusagilt infot, et antud roomamistulemit oleks plaanis linkandmetena jagada.

2.2 Internetis leiduvate andmete töötlemine

Käesoleva töö eksperimendis on jälgitavateks lehtedeks valitud Eesti avaliku sektori veebilehed, kus nagu enamus teistel veebis olevatel avalikel lehtedel, ei ole tihtipeale info lihtsalt töödeldav ja standardiseeritud vormingus.

Avaliku sektori andmete lihtsalt töödeldavale kujule viimise probleemi on üritatud lahendada erinevat moodi. Näiteks on läbi aegade välja töötatud erinevaid ontoloogiaid, mida kasutades saaks avaliku sektori asutused toota standarditele vastavaid linkandmeid ja seejärel on üritatud sisutootjate poolt juurutada nendele ontoloogiatele vastavaid süsteeme. Sellel lähenemisel on mitmeid probleeme. Esiteks toodavad avaliku sektori asutused niivõrd erineva iseloomuga sisu, et ühtsete kõikehõlmavate ontoloogiate väljatöötamine osutub ülimalt keeruliseks. Teiseks on veebis juba olemas suur hulk samade sisutootjate poolt enne vastavate ontoloogiate välja mõtlemist toodetud andmeid, mille tagantjärgi semantilisele kujule viimine ja vastloodud ontoloogiate järgi struktureerimine on väga kulukas. Kolmandaks on vaja sisutootjaid koolitada reeglitele vastavat sisu tootma ning see võtab aega ja raha. [8]

Osasid nendest probleemidest adresseeriti töös [8], kasutades andmete struktureerimiseks Wiki lehtede tehnoloogiaid. Nii saadi lahti vajadusest spetsiifiliste ontoloogiate järele ja kadus sellega kaasnev keerukus. Lisaks on Wiki lehtede loomiseks kasutatav tehnoloogia lihtsasti õpitav. Küll aga ei käsitle selline lähenemine probleemi, kus juba olemasolev mittesemantiline andmemassiiv on vaja semantilisele kujule viia ning see tugineb siiski tugeval juurutusprotsessil. Selle jätkusuutlikkus ja edu on jäigas sõltuvuses sisutootjate motivatsiooniga andmeid semantilisele kujule viia. Sellest tulenevalt oli käesoleva töö raames vaja vaadata muid lahendusi, kuidas mitte standardiseeritud kujul olevaid andmeid veebist automaatselt kätte saada.

Georgia tehnoloogiainstituudi töörühm on 2000. aastast saadik arendanud suuremahulist veebimonitoorimise lahendust nimega WebCQ. Kasutaja vaatest on see veebileht, mille sees olev raam täidab veebilehitseja funktsiooni. Selle veebilehitsejaga saab minna huvipakkuvale lehele ja kasutajaliidese kaudu saab WebCQ-le öelda, mis sorti muudatus sind täpsemalt huvitab, millist sündmust ta jälgima peaks ja kuidas sa selle kohta infot soovid (e-mailile, veebilehele, telefonile jne). Jälgimisülesannete püstitused on WebCQ sees kirjeldatud nende enda välja mõeldud

süntaksiga struktuurides, mida nad nimetavad valvuriteks (ingl *sentinels*). WebCQ sees on süsteemi optimeerimiseks ja mõttetute topeltpäringute vältimiseks kasutusel puhverserver, mis tuvastab, kas on mõtet minna uut päringut tegema näiteks selle järgi, kas võrreldes viimase päringuga on huvipakkuva lehe kontrollsumma või HTTP *Last-Modified* päis muutunud. Seda, kas huvipakkuv objekt lehel on muutunud, tuvastavad nad veel eraldi ja vastav tuvastusmetoodika sõltub valvuri tüübist. Muutus võib olla näiteks URL muutumine hüperlingis või regulaaravaldisega tuvastatud tekstiosa muutumine. Keerulisemate objektitüüpide jaoks on ka keerukamad strateegiad. Jutuks olev lahendus tundub olevat üpriski tõhus ning kuna see on olnud arenduses väga pikka aega suhteliselt suure töörühma poolt, siis usutavasti on seda läbi aja ka tugevalt optimeeritud. Negatiivseks pooleks on see, et lõppprodukt ei ole üldlevinud linkandmete standardile vastav ning ei ole ka lihtsalt liidestatav muude süsteemidega. Teiseks probeemiks on, et nende kodulehe ja avaliku repositooriumi andmetel pole mitu aastat selle projekti peal aktiivset arendust toimunud. [9][10][11]

Tasmaania avaliku sektori veebi uurimise töös [3] on välja toodud ja ka mujalt internetist otsides leiab mitmeid veebirakendusi (*WatchThatPage*, *Wisdomchange* jt), mis tegelevad info muutumise jälgimisega kasutaja valitud veebilehtedel. Kõigil selliste lehtede, mis käesoleva töö autor internetist leidis ühine probleem on, et nad ei tekita linkandmeid ja ei ole lihtsalt seotavad teiste rakendustega.

Kui algallikad ei ole lihtsalt masintöödeldaval kujul, siis veebilehtedelt info ekstraheerimine ja sellisele kujule saamine on probleem, mida on ka mitmete initsiatiivide raames juba lahendatud [12][13]. Näiteks initsiatiiv nimega import.io on spetsialiseerunud just info ekstraheerimisele veebilehtedelt. Nende baasrakendus on allalaetav programm, mille sisse on integreeritud veebilehitseja. Selle veebilehitsejaga saad liikuda huvipakkuvale veebilehele ja hiirega näidata, milliste kastide sees olev info sulle huvi pakub. Kasutaja poolt õpetatud mustrite põhjal koostatakse strateegia, mille alusel roomatakse ainult valitud lehti või minnakse mööda linke edasi ja üritatakse sama mustri alusel ka sihtkohti roomata. Saadud info saab kätte sealt samast töölauarakendusest ja lisaks genereeritakse ka unikaalne URL, mille abil saab huvipakkuva info kätte näiteks JSON formaadis (väljundformaati saab valida). Nii ei pea infot tarbivad programmid eraldi vaeva nägema, et HTML-i ja muud mitterelevantset eemaldada. Roomatakse

lehti siis, kui töölauarakendusest roomamine kliendi poolt käivitatakse. Esmasel vaatlusel jäi selgusetus, millised on võimalused roomamise automatiseerimiseks. Eraldi huvitav roomamisstrateegia oli käesoleva töö kirjutamise hetkel neil erinevate lehtede otsingumootorite tulemuste töötlemiseks. Nimelt sai läbi nende töölaua rakenduse teha huvipakkuva lehe otsingumootoris päringu ja õpetada rakendust otsingu tulemist huvipakkuvaid osi ekstraheerima. Nii sai genereerida liidese, mille kaudu sai huvipakkuva lehe otsingumootori vastu suvalise sisendiga päringu teha ning vastuse sai endale sobivas formaadis. See tähendab, et kui tarbija teadis, mis teda täpselt huvitab ja kust kohast ja kuidas seda täpselt leida, siis sai ilma eriliste valdkonnaspetsiifiliste eelteadmisteta kujundada efektiivse tõmmisestrateegia ning sai isegi lihtsalt kasutatava liidese, kust vastava strateegia alusel roomatud infot programmatiliselt kätte saada.

See kõik mida import.io pakkus oli väga efektne lahendus, aga mõningasel katsetamisel jäi mulje, et ka sellel ja teiste ettevõtete analoogsetel lahendustel oli ühiseid probleeme, mida nad ei lahendanud. Esiteks olid nad oma olemuselt kõik sünkroonsed, mis tähendas, et värskeima info kättesaamiseks pidi pidevalt järjestikuseid päringuid tegema. Teiseks olid kõik nende vastuseformaadid mõeldud pigem inimestele lugemiseks, kui automaattöötlemiseks teiste programmide poolt ehk siis need andmed ei olnud struktureeritud semantilist veebi silmas pidades. Kolmandaks ei sisaldanud vastused infot selle kohta, et kuna mingi info veebi ilmus või sealt kustus, vaid ainus info, mis tarbija sai oli see, mis viimane roomamise seis näitas.

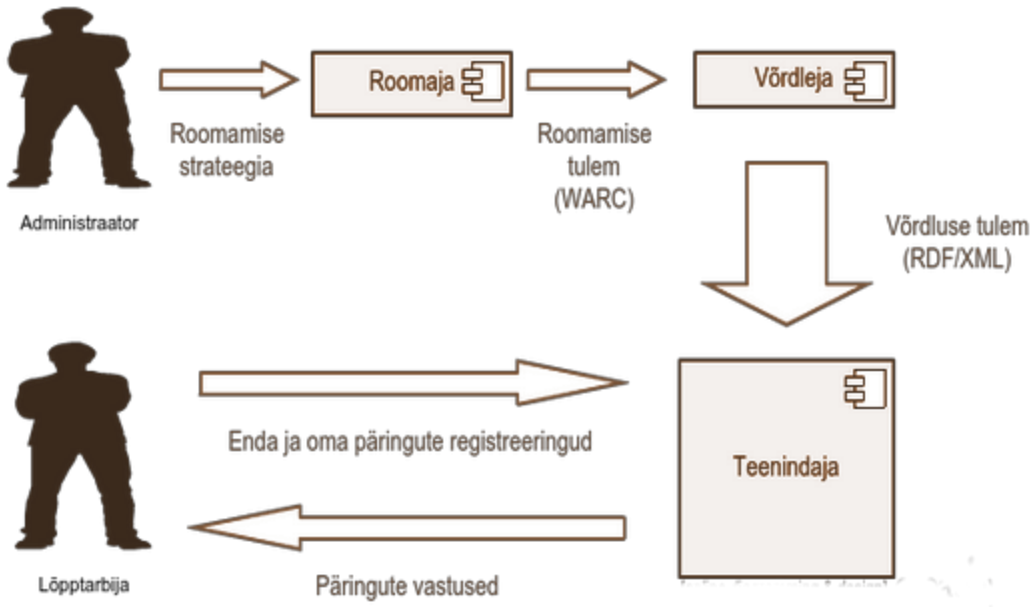
2.3 Väljundandmed käesolevas töös

Suvalistelt veebilehtedelt andmete kaevandamisega on tihti probleem, et pole teada, mis kujul täpselt need andmed on. Ükskõik kui intelligentsed ja efektiivsed algoritmid nende andmete kättesaamiseks on ei saa eeldada veatut tulemust. Sellepärast on hakatud viimasel ajal järjest enam rääkima, kuidas saaks andmete publitseerijad standardiseeritud viisil oma andmeid avaldada nii, et nad oleks lihtsamini masinloetavad ja et andmete vahele tekiksid seosed. Siit on välja kasvanud terminid semantiline veeb ja linkandmed [14]. Semantilise veebi all mõeldakse andmete esitamist selliselt, et need alluks kokkulepitud struktuurile ja oleks seega lihtsamini arvutite poolt töödeldavad (näiteks, et andmed oleks RDF/XML kujul). Linkandmete puhul on

kesksel kohal nende andmete vaheliste seoste esitamine arvutite poolt töödeldaval kujul. Selleks, et semantilise veeb juba eos läbi ei kukuks, on vaja, et andmete tootjad toodaks piisavalt linkandmeid [15]. Eestis on seis avatud andmetega küll rahuldav, aga kvaliteetsetest linkandmetest on puudus [16]. Sellest tulenevalt sai tehtud antud töös valik lõpptulem esitleda linkandmetena ja nii luua rohkem eestikeelsel infol põhinevaid kvaliteetseid linkandmeid. See on omakorda ka üks põhjustest, miks selleks ülesandeks sai komplekteeritud uus laheandus, mitte ei kasutatud mõnda olemasolevat täislahendust. Nimelt olemasolevatest lahendustest ükski ei paku väljundit, mis koosneks standardiseeritud linkandmetest.

3. Arhitektuur ja realisatsioon

Kogu käesoleva töö raames realiseeritud süsteem jaguneb suures pildis kolmeks komponendiks: veebi roomamise ja sisu talletamise komponent (edaspidi roomaja), veebi sisu muutuste tuvastamise ja vastavate andmete publitseerimise komponent (edaspidi võrdleja) ning kasutaja päringute registreerimise ja teenindamise komponent (edaspidi teenindaja). Rakenduse abstraktset kogupilti iseloomustab joonis 3.1.

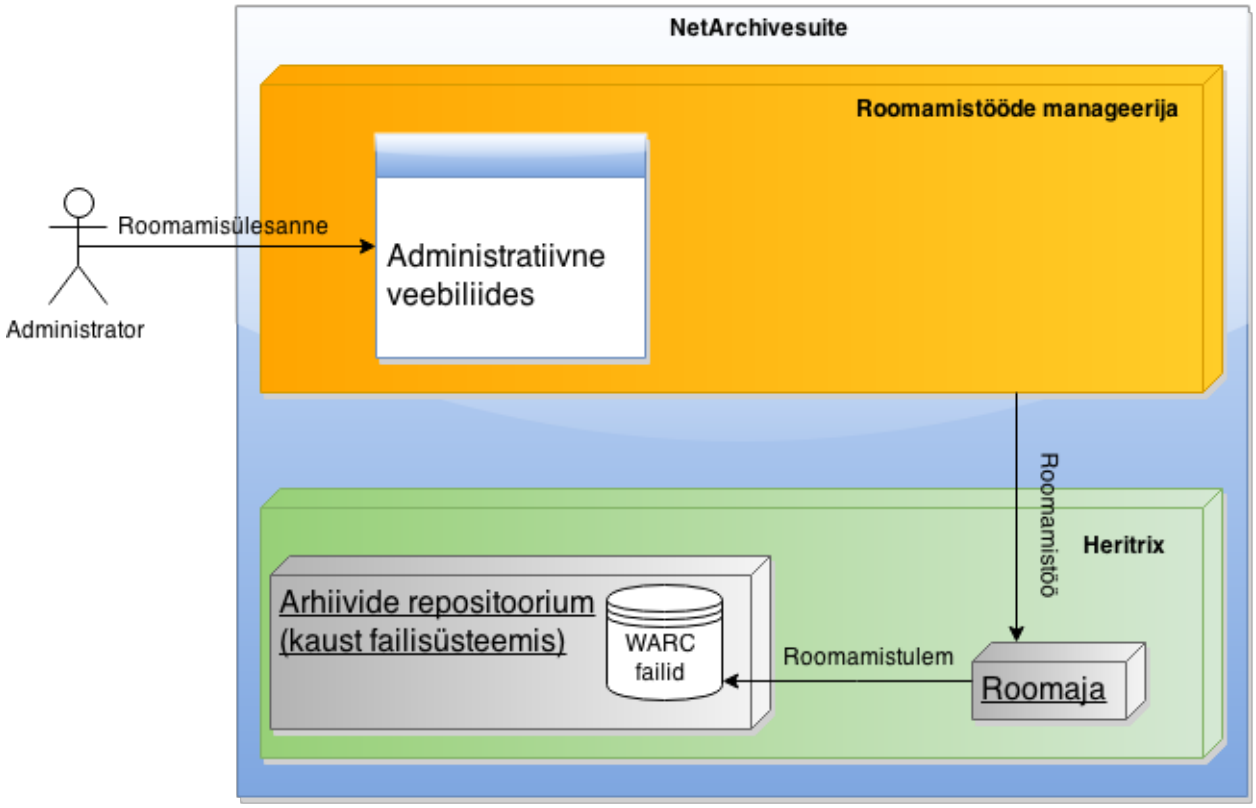


Joonis 3.1: Rakenduse üldine arhitektuur

3.1 Veebi roomamine

Roomaja administratiivliidese kaudu saab süsteemi haldaja paika panna roomamise parameetrid (valik roomamiseks sobivaid veebilehti, roomamise strateegia, maksimaalne aeg, andmemaht, periood jpm.). Roomaja rakendab etteantud strateegiat ja toodab sellest lähtuvalt roomamistulemi, milleks on selle töö raames realiseeritud lahenduse puhul veebi arhiivi (WARC) failid.

Roomajana kasutatakse lahenduses NetarchiveSuite'i. NetarchiveSuite on täieulatuslik veebiarhiveerimissüsteem, mis loodi Taani Kuningliku Raamatukogu ja Taani Rahvusliku Ülikooli Raamatukogu poolt. Selle arendust alustati 2004. aastal ja alates 2005. aastast on seda kasutatud, et taanikeelset veebi roomata. Aja möödudes on NetarchiveSuite arendusmeeskonna ja ametlike kasutajate ringiga liitunud inimesed Prantsuse Rahvusraamatukogust ja Austria Rahvusraamatukogust. [17]

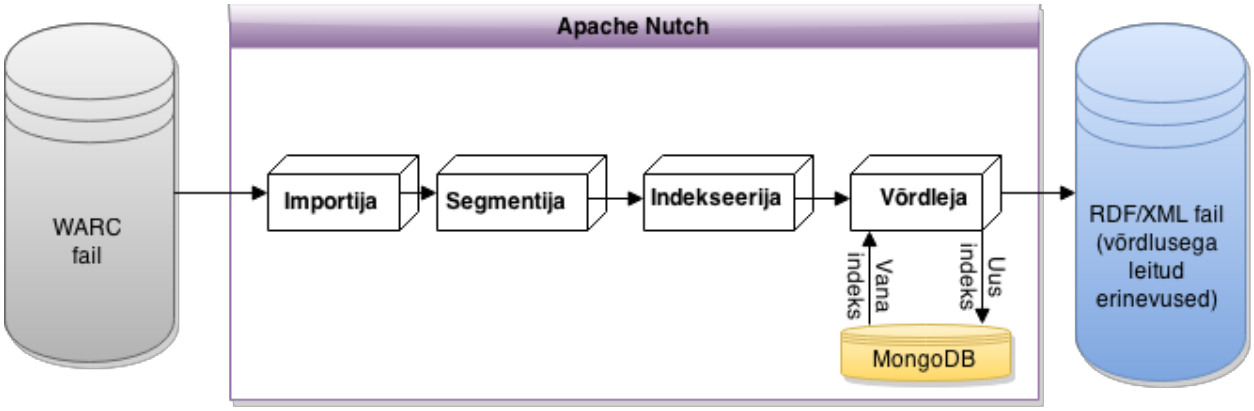


Joonis 3.1.1: Roomaja töövoog

Siit koorub välja põhiline põhjus, miks NetarchiveSuite kasuks otsustatud sai. See tarkvara on spetsiaalselt arendatud sellise töö jaoks, kus soovitakse roomata terve väiksema riigi veebi läbi. NetarchiveSuite on olnud aega, et ennast antud valdkonnas tõestada ning see on siiamaani väga aktiivses arenduses (kirjutamise hetkel on aktiivses arenduses versioon 5.0). Lisaks on NetarchiveSuite'l roomamiste seadistamiseks suhteliselt intuitiivne veebiliides. Reaalseks roomamiseks kasutab see enda sees vabavaralist roomajat nimega Heritrix. [17]

3.2 Roomamiste indekseerimine, indeksite võrdlemine ja erinevuste leidmine

Selleks, et tagada sujuv töö üleminek roomajalt võrdlejale töötab katsesüsteemis operatsioonisüsteemi skript, mis on seadistatud operatsioonisüsteemi tasandil perioodiliselt kontrollima, kas on tekkinud uusi WARC faile ja kui on, siis antud skript ehitab uute failide pealt manifesti, mille omakorda annab ette võrdlejale. Võrdlejana kasutatakse antud süsteemis Apache Nutch'i baasil loodud WARC failide töötlemiseks mõeldud komponendi NutchWAXi [18] edasiarendust [19], mida käesoleva tööga paralleelselt arendatakse samas töörühmas, aga teise bakalauresuse töö raames. See ehitab kõigepealt roomamistulemi pealt indeksi. Indeksit võrreldakse vastu eelmise perioodi indeksit ja selle tulemusena genereeritakse RDF/XML formaadis kolmikud andmete muutumise kohta. Indeks salvestatakse, et tulevikus oleks midagi, mille vastu järgmist indeksit võrrelda. Antud realisatsioonis kasutatakse indeksi salvestamiseks dokumendipõhist MongoDB andmebaasimootorit.



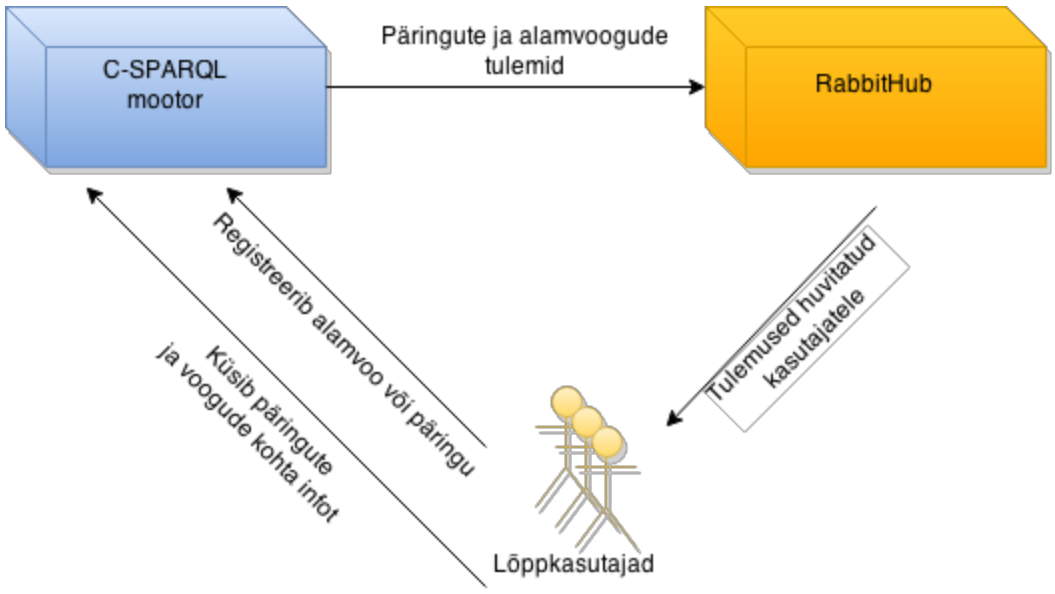
Joonis 3.2.1: Võrdleja töövoog

Võrdleja töötab kahes faasis. Kõigepealt ehitatakse NekoHTML teegiga WARC failidest leitud HTML sisust XML puu ja seejärel kasutatakse XMLUnit teeki, et teostada võrdlemist vastu MongoDB-st saadud eelmist versiooni. Siit koorub välja ka tõsiasi, et siin töös kasutatav realisatsioon tegeleb esialgu ainult HTML sisu võrdlemisega ja muus vormingus arhiveeritud andmeid esialgu lihtsalt välditakse. Kuna Apache Nutch on lihtsate vahenditega pistikprogrammidega täiendatav, siis ei ole probleem teha tulevikus realisatsioone ka teistsuguses vormingus olevate andmete jaoks [20].

Tulemina tekkinud RDF/XML kolmikute voog edastatakse kolmandale komponendile punktis 3.4.1 mainitud utiliidi abil.

3.3 Lõpptarbija päringute teenindamine

Kolmas komponent ehk teenindaja on C-SPARQL mootor, mis protsessib päringuid vastu eelmise komponendi poolt produtseeritud RDF/XML voogu ja eelnevalt teenindajas registreeritud päringuid ning alamvooge. Süsteemis on teenindaja ja lõpptarbija vahele pandud täiendav sõnumiedastusmehhanism, et tagada suuremat paindlikust ja skaleeruvust.



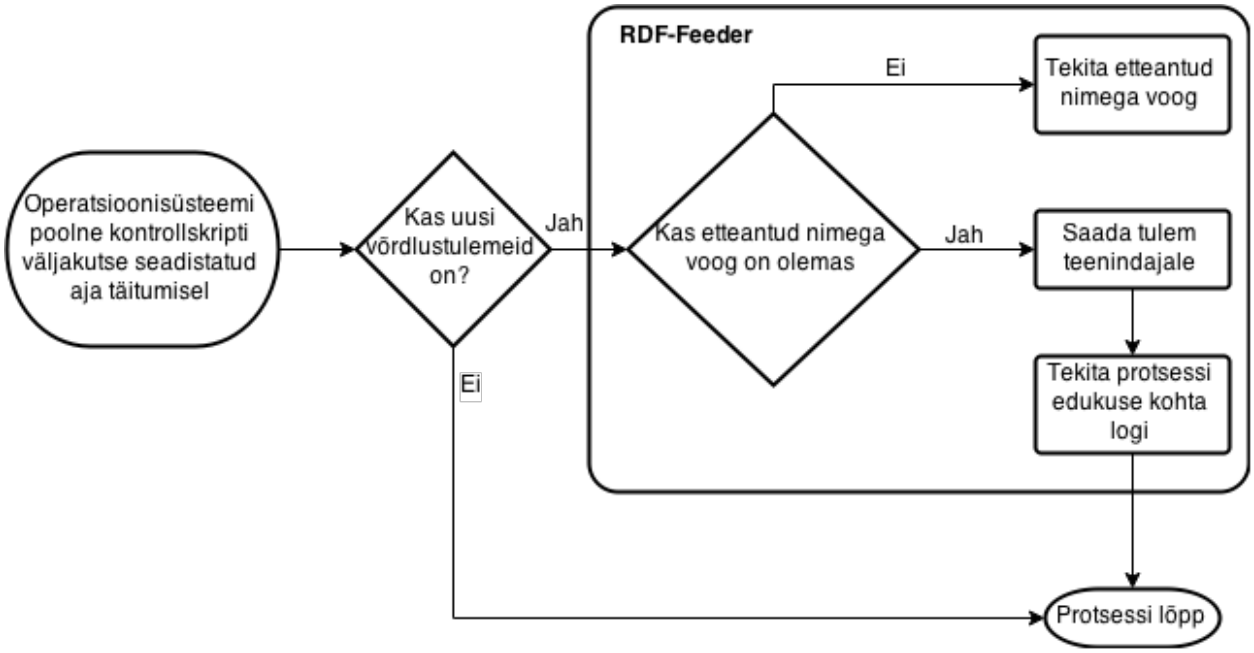
Joonis 3.3.1: Teenindaja töövoog

Lõpptarbija saab C-SPARQL mootoriga suhelda HTTP päringute abil [21]. Nii saab lõpptarbija näiteks küsida, millised päringud ja vood on hetkel süsteemis registreeritud ja selle põhjal otsustada, kas see, mida ta vajab on juba olemas ja ta peaks vastava päringu või voo tulemitele lihtsalt kuulajaks registreeruma või on vajadus tekitada uus päring või voog. Lihtpäringutele lisaks on võimalik kasutajal registreerida voog tüüpi ülempäringuid (andmehulga eeltöötlemiseks, kitsendamiseks vms) ja nende peale saab omakorda registreerida tavapäringuid.
C-SPARQL mootori realisatsiooni osas valikut ei ole. Töös kasutatav realisatsioon on ainus hästi dokumenteeritud variant. Selle realisatsiooni hea külg on ka see, et sellel on olemas hästi dokumenteeritud Java API ja sellest on aktiivses arenduses eri versioon [22], mida tehakse samas töörühmas, milles käesolevat tööd, aga teise bakalureuse töö raames. See eriversioon võimaldab suhtlust välise sõnumiedastuskihiga, mis asetseb lõpptarbija ja kliendi vahel.
Kui kasutaja otsustab registreerida uue päringu või voo, siis tuleb samanimeline olem registreerida ka sõnumiedastussüsteemis ja ennast sinna külge registreerida ning C-SPARQL mootorile tuleks vastuste edastamise lõpppunktina anda vastav sõnumiedastussüsteemi URL ning sõnumiedastussüsteemile omakorda enda rakenduse URL. Sõnumiedastajana kasutatakse antud lahenduses RabbitMQ-d ning selle peale ehitatud PubSubHubbub realisatsiooni RabbitHub. Ka RabbitHub-ga saab suhelda HTTP päringute abil. Lisaks on RabbitMQ puhul hea, et pistikprogrammina on installeeritav laiendus *rabbitmqadmin*, kust saab administraator üle veebiliidese lihtsalt jälgida sõnumiedastuse hetkeseisu ja vajadusel ka sekkuda.

3.4 Võrdleja ja teenindaja vaheline liidestus

Päringute teenindamiseks kasutatav C-SPARQL mootor võimaldab endaga suhtlemist üle HTTP. Ka uute andmete ette andmine käib HTTP abil. Selleks, et juurutamist natukene lihtsustada, on C-SPARQL mootoriga kaasas API, mis on kirjutatud Java keeles. Kuna see API ei ole iseseisvana jooksutatav vaid on mõeldud integreerimiseks teistesse süsteemidesse ja kuna võib esineda vajadus võrdleja poolt toodetud RDF/XML faili tükeldamiseks või muuks töötlemiseks, siis sai käesoleva töö raames Java keeles realiseeritud käsurealt käivitatav programm vajalike sisendvoogude tekitamiseks ja nende peale andmete saatmiseks.

Veel üks antud programmi kasulik faktor on see, et see tekitab logi, kas võrdluse ette andmine võrdlejale oli edukas ja kui ei olnud, siis mis põhjusel. Hetkel küll veaolukordade põhjalikke käitumisstrateegiaid ei realiseeritud, kuid see on tänu liidestusprogrammile lihtsate vahenditega võimalik.



Joonis 3.4.1: Võrdleja ja teenindaja vahelise liidestuse töövoog

Programmi välja kutsumiseks tuleb teha operatsioonisüsteemi kronoloogiline töö, mis kontrollib, kas uusi võrdlustulemeid on tekkinud ja kui on, siis kutsub söötjaprogrammi välja, andes talle ette uue võrdlustulemi ja vastava sisendvoo nime. Kui sisendvoogu käivitamise hetkel veel olemas ei ole, siis see tekitatakse programmi poolt automaatselt. Selle programmi realisatsioon on leitav tööga kaasas olevate lisade hulgast.

3.5 Lõppkasutaja liidestus teenuse külge

Lõppkasutaja rakendus (edaspidi LKR) peab omama ligipääsu veebile, sest suhtlus käesolevas töös realiseeritud lahenduse komponentidega käib üle HTTP protokolli. LKR saab suhelda lahenduse kahe komponendiga, milleks on C-SPARQL mootor ja RabbitHub sõnumiedastaja. Selleks, et registreerida uusi C-SPARQL päringuid ja vooge ning olemasolevate päringute ja voogude kohta infot saada, saab LKR teha HTTP päringuid C-SPARQL mootori REST liidese

vastu. Näiteks selleks, et saada infot aktiivsete päringute kohta, peab LKR tegema HTTP GET tüüpi päringu *http://c-sparql-engine-endpoint/queries* URLle ning vastus on JSON nimekirjaga aktiivsetest päringutest ja nendega seonduvast infost. Enne uue päringu registreerimist peab kasutaja looma RabbitHub'i ka vastava teema, mis käib samuti üle HTTP. Seejärel saab saata C-SPARQL mootorile päringu registreerimise taotluse ja registreerimisinfosse panna juurde ka RabbitHub teema nime, mille peale vastuseid saatma hakatakse. Kui kasutaja soovib vastuseid kuulata, siis peab ta ette valmistama HTTP lõpppunkti URLi, kuhu vastuseid saatma hakatakse ning selle URL kuulajana sõnumiedastajas ära registreerima. Nüüd igakord, kui C-SPARQL mootorile saadetakse andmed uute muudatuste kohta, lähevad need andmed läbi registreeritud päringute ja kui tekivad vastused, siis saadetakse need edasi vastavatele RabbitHub teemadele. RabbitHub vaatab seejärel, kes on vastava teema kuulajaskond ja saadab igale aktiivsele kuulajale edasi C-SPARQL mootorist tulnud päringute tulemid. Nii ei pea lõppkasutaja pidevalt kontrollima, kas on uusi vastuseid tekkinud vaid rakendus annab ise teada, kui midagi uut tuleb. Teine hea külg sellisel lahendusel on, et iga kasutaja saab vaadata, milliseid päringuid juba olemas on ja ennast kuulajaks registreerida. Lisaks on võimalik registreerida "voog" tüüpi päringuid, mille tulemi peale saab omakorda uusi päringuid teha.

Selleks, et antud töö eksperimenti läbi viia, realiseeriti ka selle töö raames LKR. See rakendus täidab ka näidise eesmärki, mille järgi teised kasutajad ennast veebis olevaid muutusi jälgiva lahenduse külge liidestada saavad. Rakendus laseb registreerida uue päringu, registreerib ise vastava teema ning rakenduse enda automaatselt kuulajaks ka. Lisaks laseb rakendus hallata olemasolevaid päringuid ja teemasid ning uurida päringute tulemeid.

Rakendusel on kaks vaadet: päringute haldamise vaade ja tulemuste vaade. Päringute haldamise vaade jaguneb kolmeks osaks. Kõige üleval on nimekiri aktiivsetest voogudest ja võimalus uus päring registreerida. Selle all on nimekiri kõikidest C-SPARQL mootoris olevatest päringutest (ka nendest, mis ei ole läbi selle rakenduse registreeritud). Selle all omakorda on nimekiri nendest päringutest, mida antud rakendus kuulab, kas hetkel või on kuulanud kunagi varem. Iga päringu juures on variant kustutada selle lokaalne info ja samuti on võimalus see kustutada C-SPARQL mootorist. Samuti on iga lokaalse teema juures nupp, mis viib tulemuste vaatele. Tulemuste vaade koosneb kahest osast. Üleval on graafik, mis visualiseerib kui palju tulemeid on

aja jooksul sisse tulnud ja all on tabel tulemite endiga. Võimalik on reguleerida graafiku intervalle ja seda, mitut uusimat tulemit all tabelis näidatakse.

Rakendus on realiseeritud kasutades modernseid tehnoloogiaid ning seda on üritatud nii teha, et see oleks hiljem lihtsalt laiendatav. See koosneb kolmest kihist. Visuaalne pool on tehtud üheleherakendusena kasutades põhiliselt HTML'i, CSS'i ja JavaScript teeki AngularJS. Vahekiht on realiseeritud Java keeles HTTP-põhiste REST teenustena, mida visuaalne pool vastavalt vajadusele välja kutsub. Vahekihi on tehtud kasutades Spring Boot raamistiku. Vahekihi ülesandeks on suhelda lokaalse andmebaasi, RabbitHub sõnumiedastaja ja C-SPARQL mootoriga. Selle vastu on loodud hulk ühikteste, mis kontrollivad, et erinevad komponendid rakenduse sees töötaks. Kolmandaks kihiks on MySQL relatsiooniline andmebaas, mis hoiab infot lokaalsete päringute ja nende tulemite kohta. Nii tagatakse see, et kasutaja saab suvalisel hetkel vaadata, mis vahepeal teda huvitava teemaga toimunud on. Rakenduse kood ja paigaldusjuhend on tööga kaasas olevates lisades.

4. Mõõtmised ja tulemused - Eesti avaliku sektori veebilehed

4.1 Eksperimendi ja andmete tutvustus

Töö käigus võeti eesmärgiks teostada ühe kuu jooksul järjestikuseid roomamisi valikule Eesti avaliku sektori veebilehtedele ning uurida saadud erinevuste arvu juhul, kui võrrelda kõiki tulemeid ning juhul, kui võrrelda nii, et jätta osad tulemid vahele.

Vaatamata sellele, et austraallaste poolt Tasmaanias tehtud samalaadne uurimus [3] ei tooda lõpptulemina standarditele vastavaid linkandmeid, on see töö nii motiivide kui põhimõtte poolest väga sarnane käesolevale tööle. Vajadus on leida kiiremaid, efektiivsemaid ja lihtsamaid viise internetis olevatest suurtest andmemassiividest huvipakkuva kätte saamiseks. Kuna praeguses uurimisfaasis oleks olnud väga suurte andmemahtude tõttu terve eestikeelse interneti peal katsetusi teha ebamõistlikult keeruline, siis tuli valida alamhulk lehtedest, mille peal esmased eksperimendid läbi viia. Austraallaste töö oli oma katsetused teinud avaliku sektori veebilehtede peal. Avaliku sektori veebilehed on üldjuhul mahukad ja nendel uuendatakse sisu enamasti

suhteliselt tihti. See teeb need eksperimendi seisukohalt heaks valikuks. Sellest tulenevalt sai ka siin töös samasugune valik tehtud. Käesoleva töö eksperiment viidi läbi 285 Eesti avaliku sektori veebilehe peal, mis on sama suurusjärk nagu austraallaste töös. Täpsem selles töös roomamisobjektidena kasutatav veebilehtede nimekiri on nähtav töö lisades. See, kuidas uurimisobjektiks valitud lehekülgi külastatakse on austraallaste ja selle töö puhul erinev. Nemad olid välja selgitanud valiklehtede esilehe URL-d ja pressiteadete lehe URL-d ning lasid enda roomajal ainult neid kahte lehte uurida iga 2 tunni järel ning lisaks võis roomaja seda aega ise ümber seadistada, kui leidis märkimisväärselt tihti või harva uusi muudatusi. Käesolevas töös ei ole valitud välja kindlaid alamlehti, vaid on roomajale ette antud esilehed ja roomaja liigub mööda neil olevaid linke vastavatel veebilehtedel laiuti ja sügavuti edasi nii kaua, kui kas kõik alamlehed on läbitud või alamlehe mahulimiit (100MB) saab täis. Lisaks on roomamised jaotatud 24 tunnisteks töödeks. Iga 24 tunni tagant käesolev töö katkestatakse vaatamata sellele, mis hetkel pooleli on ja alustatakse uue tööga, kuhu on ette antud ainult need lehed, milleni eelmine kord ei jõutud. Viimast tehakse selleks, et vältida olukordi, kus roomaja on kinni jäänud roomaja lõksu ja ei liigu seetõttu mitte kunagi edasi. Roomaja töötas korraga kuni viiekümnel lõimel. Lähtuvalt sellest strateegiast ja testmasina võimsusest võttis täis ringi peale tegemine aega 3 ööpäeva. See tähendab, et ühe tõmmise ajaline suurus siin eksperimendis ongi kolm ööpäeva ja võrreldakse omavahel erinevaid "kolmpäevakuid". Roomamisstrateegia täpsema häälestamisega on kindlasti võimalik osade olukordade puhul ühe tõmmise ajalist suurust vähendada. Selleks tuleks austraallaste töö näitel teha erinevate strateegiatega samaaegseid roomamisi erinevatele veebilehtede hulkadele ja nende osadele.

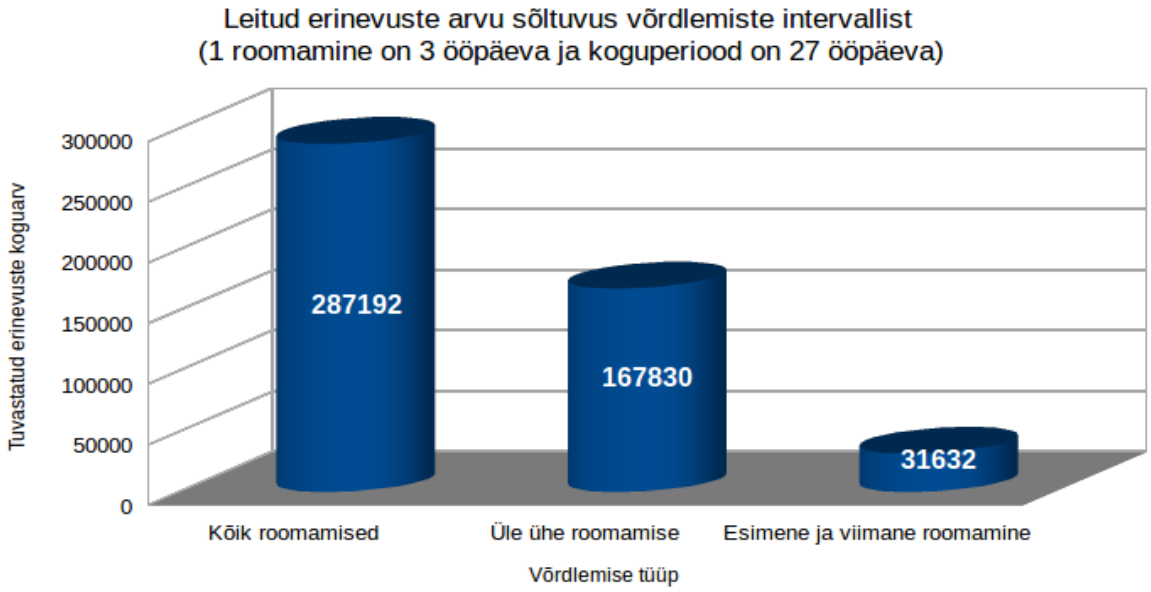
Eksperimendis oli alusmaterjaliks võetud 1. veebruarist kuni 27. veebruarini tehtud kolmepäevased täisroomamised, mida oli kokku 9 tükki. Ühe roomamise tulemusena tekkis keskmiselt 215 WARC faili, mis igaüks on keskmiselt 95MB suur. Ühe roomamise kogumaht oli siit tulenevalt natukene üle 20GB. Neid roomamisi võrreldi kolme eri moodi (iga katset alustati tühja andmebaasiga):

1.) võrreldi järjest kõiki täisroomamisi,
2.) võrreldi üle ühe olevaid täisroomamisi,
3.) võrreldi ainult esimest ja viimast täisroomamist.

Sellise katse teostasin, et selgitada välja, kui palju muutub leitud erinevuste koguarv, kui teostada võrdlemisi pidevalt või kui jätta osa materjali vahele. Eesmärk on niimoodi välja selgitada parim ajakulu ja andmete täielikkuse suhe ning vaadata, kas leitud parima suhte parameetrite juures kogu süsteem suudab tõrgeteta töötada.

4.2 Eksperimendi tulemused

Eksperimendi tulemustest selgub, et avaliku sektori veebide puhul on järjestikused roomamised siiski parim võimalik variant. Leitud erinevuste kogumahud on visualiseeritud joonisel 4.2.1. Kui võrreldi roomamiste tulemeid nii, et jäeti iga kahe võrreldava roomamise vahelt üks välja, siis oli leitud erinevuste arv 41,6% väiksem. Kui välja jäeti 7 roomamist, siis oli leitud erinevuste arv võrreldes sellega, kui midagi välja ei jäetud juba 89% väiksem.



Joonis 4.2.1: Leitud erinevuste koguarv teostades roomamisi erinevate intervallidega

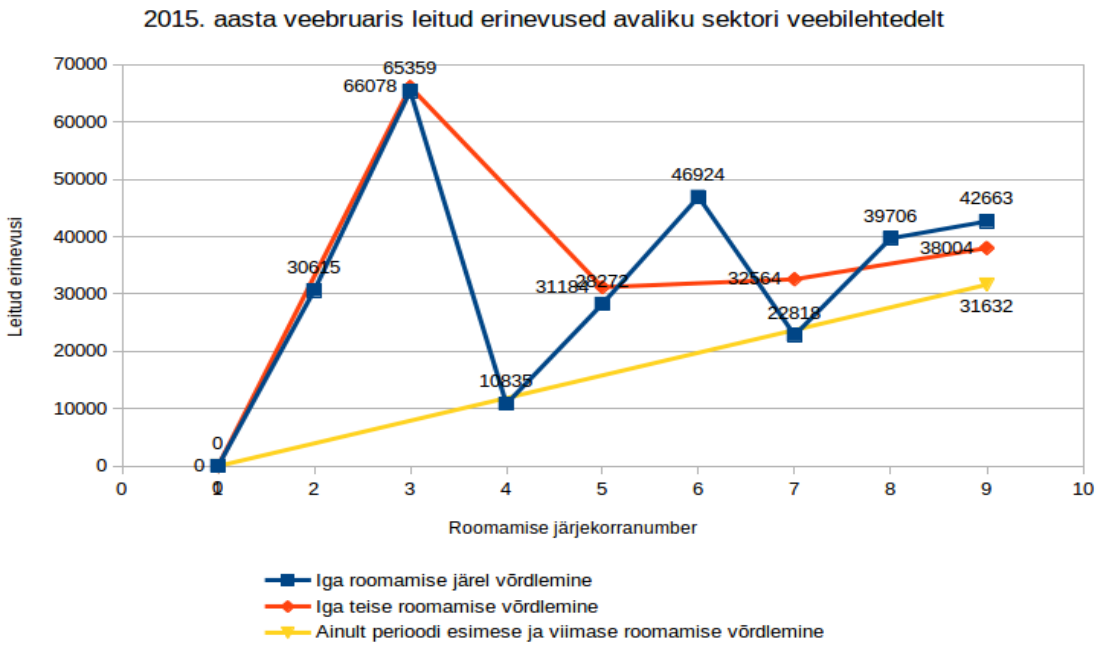
Joonis 4.2.2 näitab, kui palju erinevusi erinevate võrdluste käigus leiti. Siit leiab kinnitust sama, mis Tasmaanias tehtud töös [3], et päevad ei ole ühesugused ja erinevatel päevadel võib veebis olev sisu muutuda väga erinevas mahus. See asjaolu suurendab ka ohtu, et kui jätta roomamisi võrdlemisprotsessist välja, siis ei suudeta tuvastada suurt osa reaalselt toimuvatest muutustest.

Muid kitsaskohti selliste andmemahtude juures kogulahenduses esile ei kerkinud. Kuna roomamine võttis aega 72 tundi, siis see tähendas, et juba sellest tulenevalt võib halvimal juhul info lõppkasutajani toimetamine võtta aega vähemalt 72 tundi. Muud lahenduses olevad komponendid (nagu võrdleja, C-SPARQL mootor ja sõnumiedastaja) lisavad sellele ajale maksimaalselt mõne minuti. Tavalisel kasutusjuhul saadetakse võrdlejale roomamised ette osade kaupa iga kord, kui järjekordne umbes 100MB suur WARC fail lõpetatakse ja uus ette võetakse. Sellisel juhul võttis testmasinas roomamise võrdleja poolt töötlemine 15 sekundit kuni kaks minutit.

Joonis 4.2.2: Komponentide tööle kulunud aeg sama andmehulga töötlemiseks

Selleks, et teada saada, kuidas võrdleja suuremate andmemahtudega toime tuleb sai proovitud anda ühe korraga ette terve eksperimendi roomamistulem, mille läbi töötlemine võttis aega 27 tundi. Ehk siis ühe päeva roomamistulemi võrdlemine võtab aega kuni üks tund, mis tähendab, et võrdleja taha töövoo takerdumisest saaks rääkida alles siis, kui arhiveerija poolt peale tulevad andmemahud ajaühiku kohta oleks vähemalt kümme korda suuremad. Kahe täisroomamise tulemi võrdlemisel tekkis eksperimendis 2,1 - 14 MB RDF/XML tulemit. Kui C-SPARQL mootoris oli registreeritud 10 asjakohast päringut ja kui ühe võrdluse tulem korraga ette anti, siis kulus C-SPARQL mootoril päringute läbi töötlemiseks vastavalt võrdlusvoo suurusele 5 kuni 20

sekundit. Siit tulebki välja, et kõige suurem pudelikael antud rakenduses on roomamine ise ja antud hetkel pole mõtet muudest kohtadest, nagu võrdlemine ja päringute töötlemine optimeerida.



Joonis 4.2.3: Erinevatel hetkedel leitud erinevused erinevate roomamisintervallide puhul

Hetkel on rakendus realiseeritud nii, et tekivad väikesed seisakud andmete töötlemisel kahes erinevas punktis. Esimene selline potentsiaalne seisak tekib siis, kui veebist roomaja poolt midagi alla tõmmatakse. Nimelt ei saadeta antud infot võrdlejale automaatselt vaid jäädakse ootama seni kuni WARC faili koostamine on lõpetatud. Näiteks antud süsteemis oli roomajale pandud piir, et iga 96MB roomamise andmemahu täitumisel suletakse käsil olev WARC fail ja alustatakse uuega. Sellest tulenevalt tekitati umbes iga 18 minuti järel uus WARC fail, mis tähendab, et antud kohast võis kogu ahelas tulla 18 minutit viivitust. Seda viivitust saab leevendada vähendades uue WARC faili avamise aega, kuid 18 minutine viivitus on siin töös käsitletava informatsiooni puhul marginaalne. Teine seisaku koht on võrdleja ja päringuteenindaja vahel. Põhjus on sama, mis eelmise seisaku puhul. RDF/XML tulem saadetakse edasi alles sel hetkel, kui kogu fail on genereeritud. Antud rakenduses oli mõistlik

võrdlejale WARC faile ette anda ükshaaval, sest võrdlemine oli kordades kiirem protseduur kui roomamine (18 minuti roomamise võrdlemine ei võtnud kunagi aega üle 2 minuti). See tähendas, et ei saanud kunagi tekkida järjekorda roomaja ja võrdleja vahele. Sellest tulenevalt tuli siit juurde andmete lõpptarbijani liikumise ajale kuni 2 minutit. Päringu teenindaja suutis päringu teenindada sekunditega ka juhul, kui töötlemiseks saadeti sekundite jooksul sadu tuhandeid võrdlusi. Näiteks eksperimendi mõttes saadeti teenindajale 10 sekundi jooksul terve kuu leitud erinevused (64,8 MB RDF/XML sisu, mis on leitud üle 180GB roomamistulemite läbi töötlemisel) nii, et antud voo peale oli registreeritud kümme erinevat paralleelset päringut. Päringu tulemid jõudsid lõpptarbijani 25 sekundiga, mis tähendab, et ka sellisel ekstreemsel juhul oli C-SPARQL mootori ja RabbitHub pool tulenev viivitus ainult sekundites.

Eksperimentide järeldusena võib öelda, et kui veebilehed, millelt infot soovitakse on väga kiirelt muutuva infoga (näiteks kajastavad börsiindekseid vms), siis antud rakendus sellisel kujul info jälgimiseks ei sobi. Selleks, et niisugustele infoallikatele rakendust kohandada, tuleks tugevalt optimeerida roomamisstrateegiat ja optimeerida ka seda, kuidas andmeid mööda ahelat edasi liigutatakse. Kui aga huvipakkuvatel veebilehtedel info nii kiirelt muutuva iseloomuga ei ole, siis sobib antud rakendus muutuste jälgimiseks küll.

4.3 Ohud lahenduse töö valiidsusele

Eksperimendist selgus, et hetke kõige tõenäolisem oht lahenduse tööle ja seega ka lahenduse suurim nõrkus on roomamisstrateegia. Esiteks võib olla 72 tundi ka avaliku sektori veebilehtedele, mis oma iseloomult väga kiirelt muutuva infoga ei ole, liiga suur viivitus. Teiseks, kui vaadata nädalase ja igapäevase intervalliga roomamiste ja võrdlemiste graafikut, siis on näha päevi, kus leitakse palju erinevusi ja päevi, kus leitakse väga vähe erinevusi. See tuleneb taaskord roomamisstrateegiast mis on viimistlemata, mille tagajärjeks võivad olla roomamised, mis takerduvad veebilehtedesse kinni ja jäävad seetõttu väga ebatäielikeks. Siit saab taas tõdeda, et andmete täielikkuse esmane võti on hea roomamisstrateegia.

Siit ma järeldan, et edasine prioriteet on leida mooduseid, kuidas optimeerida roomamisstrateegiaid, sest potentsiaalselt saab tuvastada veel rohkem muutuseid, aga praegune roomamisstrateegia ei võimalda enam kiiremini algallikatele ringi peale roomata.

Veel ohte rakenduse tööle võib tulla sisu kvaliteedist. Praegusel kujul võrreldi sisu nii, et HTMLi sealt ümbert ära ei puhastanud. Kuigi lõpptulem näitab selgelt, kus täpselt sisus muutus oli, võib siiski tekkida asjatult palju erinevusi sellest, et lehel muutub struktuurne osa (HTML, JavaScript jne), mitte sisu ise.

Viimane oht on, et hulk muutusi võib jääda tuvastamata, kuna hetkel oskab lahendus ainult tekstilist sisu töödelda. See on aga NutchWAX pistikprogrammide abil järk-järgult lahendatav.

Kõige tõsisem probleem on kokkuvõttes ikkagi roomamisstrateegia ja siin suunal peaks antud lahendusega jätkama.

5. Kokkuvõte

Käesolevas töös käsitleti probleemi, kuidas järjepidevalt monitoorida suurel hulgal domeenidel toimuvaid muutusi. Selle jaoks realiseeriti lahendus, mis kasutades NetArchiveSuite veebiarhiveerijat teostab järjestuseid arhiveerimisi. Need antakse edasi sisu võrdlejale NutchWAX, mis võrdleb arhiive vastu uusimat enda andmebaasis olevat seisu ja genereerib leitud erinevuste põhjal RDF/XML voo, mis omakorda antakse edasi C-SPARQL mootorile. Sinna mootorisse saavad kasutajad registreerida päringuid ja läbi RabbitHub sõnumiedastusmehhanismi saavad kasutajad oma veebirakenduse ka nende päringu vastuste kuulajaks registreerida. Lisaks realiseeris töö autor ka näidisrakenduse lõpptarbija vaates, läbi mille saab registreerida uusi päringuid ning vaadata nende vastuseid.

Kogu lahendust katsetati valiku Eesti avaliku sektori internetilehtede peal. Eksperimendi käigus selgus, et selle jaoks kasutatava masina peal kulus etteantud strateegiaga algallikate ühekordseks arhiveerimiseks kolm ööpäeva. Prooviti teostada võrdlemist nii, et võrreldakse kogutulemit ja jäetakse osad võrdlused vahelt ära. Eksperimendist selgus, et tulemite vahelt välja jätmine ei ole antud süsteemis mõistlik, kuna sel juhul võib kahaneda leitud erinevuste arv kordades. Lisaks selgus ka, peale arhiveerimise süsteemi teiste osade käitamine võttis kogu süsteemi mõttes aega tühiselt vähe ja ka sellest tulenevalt ei oleks mingit võitu, kui roomamist harvem teostada. Samuti oleks kogu erinevuse tuvastamise ja selle info lõpptarbijani viimise mõttes vahe marginaalne, kui teha töökiiruses võitu andvat optimeerimist päringute teenindaja või arhiveerimiste võrdleja osas. Jõuti selgusele, et pigem tasub sellise süsteemi puhul esialgu rõhku panna effektiivsetele roomamisstrateegiatele.

Summary

Architecture of a Estonian-domain continuous Web monitoring system

Tarmo Kalling

Given work adressed the problem, how to continously monitor content changes in a large set of web domains. In order to accomplish that a solution was implemented which uses NetarchiveSuite to perform continous archivings of the same domains. The result of those archivings is passed on to NuchWAX which compares the webcontent to its own database and produces RDF/XML stream from the differences found. This RDF in turn is fed to a C-SPARQL engine. End consumers can register their queries in that engine and with the help of RabbitHub messaging system, users can register their web applications to listen the results. In addition to that, author of the given work implemented an example web application through which one can register new queries and query result listeners.

This whole solution was tested on a variety of Estonian public sector web domains. From the experiment it occured that it would take three days to archive the given set of domains once. It was experimented to leave some of the archives out from the comparsion process to see how this affects the results. It occured that given the used crawling strategy, it is not reasonable to leave any of the archives out from the comprsion as the count of found differences can fall multiple times. It also occured that other parts besides the archiveing took minimal time in the context of the whole process, so we would gain nothing from cuting down on archivings. The upsides from optimizing NutchWAX, C-SPARQL engine, RabbitHub or the components between them would be minimal in the given moment so the recommendation for future is to put effort in archiveing strategies.

Tänuavaldused

Soovin tänada töö juhendajat, Peep Küngast, kes täitis oma rolli väga hästi ning oli alati hea nõuga abiks. Samuti soovin tänada endaga samas töörühmas olnud tudengeid Mikk-Erik Bachmanni ja Kaarel Tõnissoni, kes aitasid mind nõu ja jõuga vastavalt päringuteenindaja ja võrdleja juurutamisel.

Tarmo Kalling

Viited

[1] M. J. Miller, „Cisco: Internet Moves 21 Exabytes per Month“, 2010 http://www.pcmag.com/article2/0,2817,2361820,00.asp 02.04.2014, 20:34 (UTC).

[2] S. Abiteboul, „Issues in Monitoring Web Data”, *Lecture Notes in Computer Science*, **2453**, 2002, 1-8, DOI:10.1007/3-540-46146-9_1

[3] Y. S. Kim ja B. H. Kang, „Tracking Government WebSites for Information Integration", *Information research: an international electronic journal,* **12.4**, 2007

[4] M. Naghavi ja M. Sharifi, “A Proposed Architecture for Continuous Web Monitoring Through Online Crawling of Blogs”, *International Journal of UbiComp*, **3.1**, 2012, DOI:10.5121/iju.2012.3102

[5] W. Huang, L. Zhang, J. Zhang ja M. Zhu, "Semantic Focused Crawling for Retrieving E-Commerce Information", *Journal of Software*, **4.5**, 2009, 436-443, DOI:10.4304/jsw.4.5.436-443

[6] M. Day, “Collecting and preserving the World Wide Web”, 2003 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.388&rep=rep1&type=pdf 10.05.2015, 18:54 (UTC)

[7] Suurbritannia veebi arhiivi ametlik tuvustav lehekülg http://www.webarchive.org.uk/ukwa/info/about 11.05.2015, 21:14 (UTC)

[8] C. Wagner, K. Cheung, K. S. K. Cheung ja R. K. F. Ip, “Building Semantic Webs for e-government with Wiki technology”, *Electronic Government, an International Journal*, **3.1**, 2006, DOI:10.1504/EG.2006.008491

[9] L. Liu, W. Tang, D. Buttler ja C. Pu, “Information Monitoring on the Web: A Scalable Solution”, *World Wide Web*, **5.4**, 2002, 226-333, DOI:10.1023/A:1021028509335

[10] WebCQ ametlik veebileht http://www.cc.gatech.edu/projects/disl/WebCQ/WebCQ.html 15.05.2015, 14:12 (UTC)

[11] WebCQ koodirepositioorium http://sourceforge.net/projects/webcq/ 15.05.2015, 14:30 (UTC)

[12] Rakendus *import.io* veebilehtede roomamiseks ja saadavatest andmetest info ekstraheerimiseks. http://support.import.io/knowledgebase/articles/251955-what-is-import-io 04.03.2015, 12:12 (UTC)

[13] Veebilehitseja Chrome pistikprogramm *getdata.io*, mille abil disainida konkreetsete lehtede roomamiseks strateegiaid. https://getdata.io/ 04.03.2015, 13:14 (UTC)

[14] C Bizer, T. Heath ja T. Berners-Lee, „Linked Data – The Story So Far", *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, 2009, 205-227, DOI:10.4018/978-1-60960-593-3

[15] N. Kroes, „Embracing the open opportunity. European Commission - SPEECH/14/556", 2014 http://europa.eu/rapid/press-release_SPEECH-14-556_en.htm 10.12.2014, 16:11 (UTC)

[16] U. Valner, „Avaandmed – samm tuleviku internetti", *Eesti infoühiskonna aastaraamat 2011/2012*, 2012, 12-17

[17] NetarchiveSuite ametlik tutvustav lehekülg https://sbforge.org/display/NAS/NetarchiveSuite 02.01.2015, 16:17 (UTC)

[18] NutchWAX ametlik tutvustav veebileht http://archive-access.sourceforge.net/projects/nutchwax/ 02.02.2015, 11:48 (UTC)

[19] K. Tõnissoni poolt realiseeritud NutchWAX edasiarendus, mis teostab võrdlemist https://bitbucket.org/kaareltonisson/nutchwax-with-diffing/ 05.05.2015, 23:39 (UTC)

[20] Apache Nutchi pistikprogrammidega seonduvat tutvustav ametlik lehekülg https://wiki.apache.org/nutch/PluginCentral 05.05.2015, 22:11 (UTC)

[21] M. Balduini, E. Nitto, M. Miglierina, V. Munteanu, G. Casale, J. Pérez ja W. Wang, "C-SPARQL Engine RESTful Services Java API", *Monitoring Platform Initial Release*, 2013, 29-31

[22] M. Bachmanni poolt realiseeritud C-SPARQL mootori edasiarendus, mis võimaldab suhtlust RabbitHub sõnumiedastustarkvaraga https://github.com/a71993/csparqlpush/tree/master/rsp_services_csparql 07.05.2015, 08:12 (UTC)

Lisad

Erinevad programmikoodid ja muud tööga seotud failid on tööga kaasas oleval CD plaadil ja repositooriumites

1.) Abikomponent võrdluste ette saatmiseks teenindajale:
   CD: */rdf-feeder*
   Repositooriumis: https://github.com/Boomber/rdf-feeder
2.) Päringute administreerimiseks ja lõpptulemuse analüüsimiseks kasutatav veebirakendus:
   CD: */rdf-analyst*
   Repositooriumis: https://github.com/Boomber/rdf-analyst
3.) Roomajale ette antud allikate nimekiri, eksperimentide manifestfailid ja RDF/XML tulemusfailid on eksperimentide kaupa CD-l kaustades:
   /experiment/1/ (eksperiment, kus võrreldi kõiki arhiive)
   /experiment/2/ (eksperiment, kus võrreldi arhiive üle ühe)
   /experiment/3/ (eksperiment, kus võrreldi ainult kuu esimest ja viimast arhiivi)
   Repositooriums on info sama struktuuriga: https://github.com/Boomber/bsc-thesis-extras

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Tarmo Kalling

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   "JÄRJEPIDEVA VEEBIMONITOORIMISE SÜSTEEMI ARHITEKTUUR EESTI DOMEENIS"

   mille juhendaja on Peep Küngas

   (a) reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   (b) üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile;
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **20.05.2015**